CASE STUDY

Tarsnap's FreeBSD Cluster

BY COLIN PERCIVAL

Tarsnap is an online backup service with an emphasis on security—indeed, when I started the company back in 2006, I quickly settled on the tagline "Online backups for the truly paranoid" to reflect the fact that as a cryptographer and FreeBSD Security Officer, my aim was to provide a service which was secure enough that I would trust it with my own secrets. Tarsnap is also one of the leading reasons that FreeBSD is available in Amazon EC2: Tarsnap needed to run in EC2 (among other things, so that it had cheap and fast access to the Amazon S3 storage service), but I needed an operating system which I trusted and knew I could administer easily—in other words, Tarsnap needed FreeBSD in EC2, and I scratched my itch.

Customized AMIs

While all of the EC2 instances Tarsnap uses run FreeBSD, none are ever launched from the "stock" FreeBSD Amazon Machine Images (AMIs) which the FreeBSD project publishes. Instead, I make use of a tool I created five years ago to build lightly customized FreeBSD images: An "AMI Builder" AMI, available for FreeBSD 12.2-RELEASE as `ami-085ee41974babf1f1` in the us-east-1 EC2 region. These AMI Builders are not currently provided by the FreeBSD project but are instead something I build and publish myself after each release; at some point I hope to integrate these into the builds performed by the release engineering team.

To create a "Tarsnap FreeBSD 12.2-RELEASE" image, I start by launching the aforementioned AMI Builder—and then I wait, roughly 20 minutes, while the virtual machine spins up and installs (stock) FreeBSD 12.2-RELEASE onto its virtual disk. This disk is then mounted on `/mnt/` while the FreeBSD system mounted at / runs from a memory disk and starts an `sshd` process.

Once I can SSH into the AMI Builder (like other FreeBSD images in EC2, using the SSH key I provided to EC2 and the user name `ec2-user`), I set to work with some standard configuration which I want on all of Tarsnap's systems:

- I build and install some packages from the FreeBSD ports tree: pkg, djbdns, qmail, spiped, and tarsnap.
- I enable some daemons I want (svscan and spiped), disable other code I don't want (sendmail and firstboot pkg installation), and lock down some settings (disabling network listening in syslogd and restricting sshd to IPv4) in `/etc/rc.conf`.
- I instruct pkg to use packages I build myself instead of the packages distributed by the FreeBSD project, by creating configuration files in `/usr/local/etc/pkg/repos/`.
- I add cron jobs to run `freebsd-update cron` and `pkg upgrade -qn` every morning—I don't want updates installed automatically, but I definitely want to get an email when they're available.
- I set up djbdns to provide a local DNS cache and set `resolv.conf` to point at it.

- I set up qmail to send outgoing email via my mail server.
- I set up spiped to create secure connections to my mail and package servers, and also to wrap incoming SSH connections.

Finally, after performing all the configuration I want on `/mnt/`, I unmount the disk and ask EC2 to "create an AMI from the running EC2 instance". Despite the description of this EC2 API call, the AMI created does not reflect what was running, but rather the current state on disk—in other words, it ignores the FreeBSD system running out of a memory disk and creates an AMI corresponding to the configuration I performed on the filesystem mounted at `/mnt/`.

Now I have a configured "Tarsnap FreeBSD" image from which I can launch instances with all of my preferred defaults set up, and there's one last step: I ask EC2 to copy this AMI from the us-east-1 region to the us-west-2 region. While almost all of Tarsnap's servers run in us-east-1 (which was the only EC2 region when I launched Tarsnap) I do have one system in us-west-2: A monitoring system which alerts me if anything breaks.

pkg Builder

The FreeBSD Project provides binary packages for software in the ports tree, and most users will want to make use of those rather than building from source. For Tarsnap's servers, I do my own builds, for two principal reasons:

- In some cases, I want non-default port options.
- As a FreeBSD developer, sometimes I commit updates and want to use them immediately, rather than waiting for the next scheduled package set.

It's possible that at some point in the future neither of these will apply—it may be that the ports where I set non-default options will gain "flavors" which provide what I need, and it may be that the FreeBSD Project will someday perform shockingly fast package builds every time a port is updated (but considering the rate at which new compilers are released with ever-worsening performance, this seems unlikely). In the meantime, running my own package builds is easy and convenient.

Tarsnap's "cluster" includes two very lightly used web servers.

I use poudriere for this purpose, and the process of setting up the builds is very easy: After installing poudriere, simply `poudriere ports -c` and `poudriere jail -c -j JAILNAME -v 12.2-RELEASE`. After that, I set a few options in `/usr/local/etc/poudriere.d/make.conf`, set a cron job to run `poudriere bulk -f /root/pkgs-wanted` (where I have a list of packages I want), and use lighttpd to serve up the resulting packages.

Building a complete set of the packages I use on Tarsnap's servers takes about 2 hours on the $15/month "t3.small" EC2 instance I use, but most package build runs complete far faster than that, thanks to poudriere operating incrementally and not recompiling unchanged packages. Indeed, I would use an even smaller EC2 instance but for one detail: Some of the C++ compiles fail on instances with only 1 GB of RAM.

Web Servers

Tarsnap's "cluster" includes two very lightly used web servers, running on EC2 "t3.nano" instances. In an era of web frameworks and rich web applications making javascript function calls, the Tarsnap website is perhaps exceptional mainly for its archaic design: The public portion of the

website is entirely static HTML—hand-written aside from a shell script which wraps content with a header and navigation section—and the account creation and management code consists of a handful of CGI scripts… written in C. While CGI scripts have inherent performance problems—forking a process is far more expensive than running code within the context of the web server or forwarding requests to another long-running daemon—as far as Tarsnap is concerned, I would love to be in the position of experiencing performance problems due to an excess of customers using the website.

There are a few slightly more modern aspects of the web servers, however: TLS traffic is unwrapped using hitch, in order to keep the cryptographic code segregated from the web server code; TLS certificates are obtained using Let's Encrypt and certbot; and TLS private keys are kept in an Amazon Elastic File System (aka. NFS) filesystem. While the use of hitch, Let's Encrypt, and certbot are very common, the use of Amazon EFS probably deserves some explanation.

The use of Amazon EFS solves a bootstrapping problem: I obtain TLS certificates via the "webroot" mechanism, wherein a certificate issuing request is authorized by placing files into a `/.well-known/acme-challenge` directory on the website. This only works once traffic for the website is directed to the server in question—but I don't want to direct traffic to a new host until it is ready to respond to HTTPS requests. Storing private keys in a manner which survives the replacement of a web server instance solves this problem.

Now, NFS is not known for having a stellar track record with regard to security, and operates in plaintext, neither of which seems ideal for cryptographic material—but the fact is that the guarantees made by TLS are now sufficiently weak that no security is lost here. If an attacker can intercept traffic on the Amazon EC2 network, they can trick Let's Encrypt into issuing them a new certificate allowing them to impersonate Tarsnap's web servers—so having another attack which would rely on intercepting traffic on the Amazon EC2 network does nothing to extend their capabilities.

Mailserver

All of Tarsnap's email is routed through a single mailserver. This includes:

- "Human" emails sent between me and the outside world.
- "Logging" emails generated by cron jobs (which land in my inbox).
- Transactional emails generated when users sign up for Tarsnap, confirm their email addresses, make payments, or need to be warned that their (prepaid) accounts are running out of money.
- Public mailing lists, both for Tarsnap announcements and discussions, and for open source software which originated from Tarsnap.

For historical reasons, the Tarsnap mail server runs qmail; specifically, "this is what I started with when I configured my first FreeBSD server, nearly 20 years ago." If I were starting from scratch, I would probably use postfix, but in the long-standing tradition of sysadmins everywhere, as long as it isn't broken, I'm unlikely to fix it.

Email arrives at the mailserver via two routes: Connecting to port 25 on the external network interface and connecting to spiped via port 8025—which then arrives at qmail via port 25 on the loopback interface. Using spiped in this manner not only secures "internal" network traffic, but also neatly solves the question of email relaying: Any email which arrives at qmail via the loopback interface can be safely relayed to other domains.

Since I use qmail, I naturally use Dan Bernstein's ezmlm (and its extension, ezmlm-idx) to manage Tarsnap's mailing lists. The "tarsnap announcements" list is moderated, but the rest are open to postings from any subscriber; fortunately, I have had no problems with trolling, and thus far

spambots don't seem to be smart enough to subscribe to mailing lists before attempting to send email through them.

Tarsnap's mailing lists have archives accessible via HTTP/HTTPS. I use mhonarc to take the mailing list posts from ezmlm-idx and convert them into HTML format; lighttpd to serve them up to the world; and hitch to add a layer of TLS for those who want it. It's hard to imagine what security is added by TLS here: The mailing list posts are public, and the number of bytes transferred is enough to uniquely identify the page being downloaded; nonetheless, some people strongly prefer to use TLS even when it serves no purpose.

Finally, outgoing email is dispatched via a shell script which runs one of two "qmail-remote" programs: Either the original qmail-remote, which sends email directly via SMTP, or a "qmail-remote-ses" I wrote which sends email via Amazon's Simple Email Service. Unfortunately delivering email into inboxes is increasingly difficult, so for transactional emails I hand the job over to Amazon; at a cost of $0.10 per 1000 outgoing emails, Amazon doesn't need to be very much better at delivering email to pay for itself when it comes to customers signing up to give me money. On the other hand, people who have overly restrictive mail servers are unlikely to subscribe to Tarsnap's mailing lists in the first place—so I have no problem with relying on qmail and direct SMTP for that email traffic.

Monitoring

As mentioned earlier, while most of Tarsnap's systems are in Amazon's us-east-1 region, I have a monitoring system in the us-west-2 region. There are two reasons for having this instance in a different region from everything it is monitoring: First, to allow it to detect outages related to the AWS region's external network connectivity; and second, to minimize the likelihood that a failure disables both Tarsnap's systems and the monitoring simultaneously. Furthermore, even if an outage does knock two AWS regions offline simultaneously, it's (a) unlikely that there's anything I could do to respond to it, and (b) very likely that the world is facing far more important problems than Tarsnap being offline.

I use a handful of simple shell scripts to perform a range of monitoring tasks.

Rather than using any of the widely used monitoring frameworks, I use a handful of simple shell scripts to perform a range of monitoring tasks (pinging servers, fetching web pages, performing Tarsnap backups) from cron jobs; these each emit a state ("GOOD", "FAILED", or "TIMED OUT"), and another script uses Twilio to send SMS messages and make phone calls. The combination of a small number of notifications and Twilio's usage-based pricing makes this extremely affordable—indeed, most of the cost is the $1/month fee to rent a phone number, which is required in order to send SMS messages.

Holding Everything Together: spiped

The astute reader will have noticed that I've mentioned spiped a few times but without giving many details. Since this is a little-used tool—and one I wrote myself specifically for securing connections to and between Tarsnap systems—I think it deserves special attention.

The spiped utility, at its core, is a tool for connecting one socket address to another socket address in a cryptographically secure manner. In this sense it can be seen as a replacement for stunnel or "ssh -L" port forwarding; but whereas stunnel relies on the overly complex TLS protocol and

certificates, and ssh uses a persistent TCP connection over which tunneled connections are multiplexed, spiped uses a pre-shared secret and opens a new connection for each connection being relayed—making it much simpler and giving it the same behavior as TCP with regard to network failures. At present, spiped supports TCP sockets over IPv4 and IPv6, as well as local ("UNIX") sockets.

The name's origin, "secure pipe daemon," may give another hint to the intent, however: Whereas the introduction of pipes to UNIX in 1973 led to an explosion of functionality as utilities were combined in various ways, I wanted to develop software which used a multitude of daemons—and to be able to connect them without regard for whether they would end up running on the same host or across insecure networks.

As mentioned above, I use spiped to secure SMTP connections to Tarsnap's mailserver; I also use it to secure POP3 connections, which has the benefit of allowing me to use POP3 in its simplest and least secure mode—no TLS and plaintext passwords—while retaining the necessary security. (Hey, it works, ok? I've been meaning to switch over to IMAP for decades.)

Similarly, connections to the pkg builder are protected by spiped; while the contents of the packages are not sensitive (they all come directly from the FreeBSD ports tree) this allows me to ensure package integrity without needing the more heavy-weight option of using poudriere to sign packages and may also protect the pkg builder somewhat in the (unlikely) event that a vulnerability is found in lighttpd.

Finally, I use spiped to protect all of my SSH connections—while all of the systems I use have sshd enabled in order to allow me to easily administer them, port TCP/22 is blocked and the only way to reach sshd is via an spiped process which doesn't allow any traffic through until the incoming connection has been cryptographically authenticated.

Legacy Systems

Of course, in addition to the aforementioned infrastructure, there's the Tarsnap backup service itself. This runs on what one might call "legacy" systems: Since it is directly responsible for ensuring the safety of customer data—not to mention bringing in revenue—I allow the backup service to lag behind other systems (except where security is concerned, naturally). At present it uses an older version of FreeBSD on an older EC2 instance type—a fact which protected it from the stability problems in the (recently added) ENA network interface driver which were corrected in the FreeBSD-EN-20:11.ena Errata Notice—and the only third-party packages installed are those which I have in my "Tarsnap FreeBSD AMI"—i.e. those needed to allow me to securely connect for administration purposes, and those used to send outgoing email. Aside from those, the only code running is Tarsnap's proprietary code—a few daemons, plus cron jobs for internal performance monitoring and nightly billing purposes.

Over time I expect to use more open-source code in the backup service.

Over time I expect to use more open-source code in the backup service–or perhaps I should say, I have released open source code which I expect to be using in the backup service in the future. I designed the kivaloo data store around Tarsnap's needs for metadata storage, and it only made sense to release the code first and use it later. Tarsnap stores customer data in Amazon S3, but it's necessary to keep track of where each data block was stored in order to be able to retrieve it on demand. This leads to a data store usage pattern—tables with keys and values of roughly 40 bytes each—for which most data stores are poorly optimized.

CASE STUDY

Future Directions

I can't conclude without mentioning something Tarsnap isn't using yet, but which I'd like to use: "Graviton 2" arm64 EC2 instances. In all of my testing to date these have performed extremely well—and they're roughly 40% less expensive than the x86 EC2 instances currently used by Tarsnap.

Naturally, while FreeBSD images are available for arm64 EC2 instances, Tarsnap won't be making use of them (aside from development and testing) until the arm64 architecture is fully supported by the FreeBSD Project—including having binary updates available via FreeBSD Update. I understand that this is likely to arrive in time for FreeBSD 13.0; so, by this time next year I may have replaced many systems with new arm64 instances. (The core backup service, of course, will continue to run on x86 for a while longer.) Only time will tell, but it's safe to say that this is an area where I'm very excited about the future.

COLIN PERCIVAL has been a FreeBSD developer since 2004 and was the project's Security Officer from 2005 to 2012. In 2006, he founded the Tarsnap online backup service, which he continues to run. In 2019, in recognition of his work bringing FreeBSD to EC2, he was named an Amazon Web Services Hero.